QUARKUS

Java
Rest Services
Bean Validation

JEE - Microservices

@ CGS IT - 2023

Inhalt

- Overview
- Bean Validation Annotations
- Example
- Manual Bean Validation
- Custom Bean Validation Annotation

Bean Validation - Overview

- The Bean Validation Framework us used to validate Data by using Annotations or Validation-Methods or Classes to ensure valid objects
- This is usually done in Rest-APIs or for JPA Entities to be validated when received or stored.
- Validation can also be used programmatically. Furthermore, it can be used to ensure valid Method input or return Values
- The Bean Validation APIs and annotations are defined in the dependency Jakarta.validation package

Bean Validation - Annotations

Annotation	Description
@Constraint	Marks an annotation to be a bean Validation Constraint
@Valid	Marks a method parameter, return type or property for validation
@Null - @NotNull	Null or not Null Validation
@NotBlank	The annotated element must not be null and must contain at least one non-whitespace character
@Min - @Max	Used for numbers to validate lower or equal or higher or equal to the specified value
@DecimalMin, @DecimalMax , @Digits	The annotated element must be a number whose value must be lower or equal to the specified maximum or minimum.
@Size	The annotated element size must be between the specified boundaries (included).
@Furture - @Past - @FutureOrPresent	The annotated element must be an instant, date or time in the future/past
@AssertTrue - @AssertFalse	The annotated element must be true. Supported types are boolean and Boolean. null elements are considered valid !
@Pattern	must match the specified regular expression. The regular expression follows the Java regular expression conventions see java.util.regex.Pattern.

Bean Validation Example

- Usage Examples can be found in the Example Repository package
  “at.cgsit.jeemicro.bean_validation”

```
public class BVTestObject {

   @NotEmpty(message = "name may not be empty")
   @Pattern(regexp = "[a-zA-Z0-9]*",
         message = "name must be alphanumeric")
   private String name;
   @Future
   private Calendar futureDate;
   @Email
   private String email;
   @Min(0)
   @Max(100)
   private int percent;
   @PositiveOrZero
   @Digits(integer = 5, fraction = 2) // 5 digits in total
   private BigDecimal amount;

   @AssertTrue(message = "this chat message is not allowed.")
   public boolean isChatMessageAllowed() {
      if("chris".equalsIgnoreCase(this.name)) {
         return false;
      } return true; }
```

CGS ITSolutions

Bean Validation – Use Validator

- Using the validator is simple
- Inject the Validator to your code via CDI
- Call validator.validate() for your objects
- Verify if there are any ConstraintViolations found

```
@QuarkusTest
class BVTestObjectTest {

    @Inject
    Validator validator;


@Test
void isChatMessageAllowed() {
    Set<ConstraintViolation<BVTestObject>> validate
            = validator.validate(createTestObject());


    validate.forEach(v -> System.out.println("failed: " +
v.getPropertyPath() + " message: " + v.getMessage()));
    assertEquals(2, validate.size());
}
```

Bean Validation - Cascading

- Basically, Bean Validation does not cascade into other POJO Relations
- If @Valid Annotation is used on a referened single or List Object,
- the validation is cascaded to those referenced objects also

```
@NotNull @Valid
private BVTestObject2 referenceObject;
```

“quarkus.hibernate-validator.fail-fast“
When fail fast is enabled the validation will stop on the first constraint violation detected.

CGS ITSolutions

Danke für Ihre Aufmerksamkeit

CGS ITSolutions